TRACEINO

Traceino Users Guide

Version: DRAFT: 0.2 corresponding to Traceino Release 1.0.0

Date: 04. December 2013

Web: www.modelbus.org

Email: info (at) modelbus (dot) org

Fraunhofer
FOKUS

MOTION
Modeling and Testing for
Systems and Service Solutions

This document has been created by the ModelBus group at Fraunhofer FOKUS (www.modelbus.org).

1. About Traceino

Traceino is a traceability framework for model driven development based on the Eclipse Modeling Framework (EMF). In a unique way, Traceino allows the definition and utilization of type safe case specific traceability links without losing the important generic characteristics necessary for analysis like change impact analysis and coverage analysis.

The Eclipse update site of Eclipse contains a set of features to ease the development of traceability meta models with the aid of standard EMF editors (tree editor as well as the diagram editor) and a complementary traceability specific view for easily modeling traceability specific aspects. For example, Traceino integrates tools like Topcased OCL (see http://www.topcased.org/index.php?id_projet_pere=30) or Epsilon EVL (see http://www.eclipse.org/epsilon/doc/evl/) for capturing constraints for the model elements to be linked by a specific traceability link type.

Another aspect of Traceino is its independence of tools. The framework is designed to be integrated in any tool needed in a particular development process. The tool integration specific libraries of Traceino are implemented based on Java and SWT for the Traceino specific UI dialogs which leads to a high degree of flexibility. In addition, a .NET implementation of the Traceino framework is available. Traceino is shipped with a set of basic implementations for tools based on Eclipse which limits the implementation effort for those tools to a few lines of code.

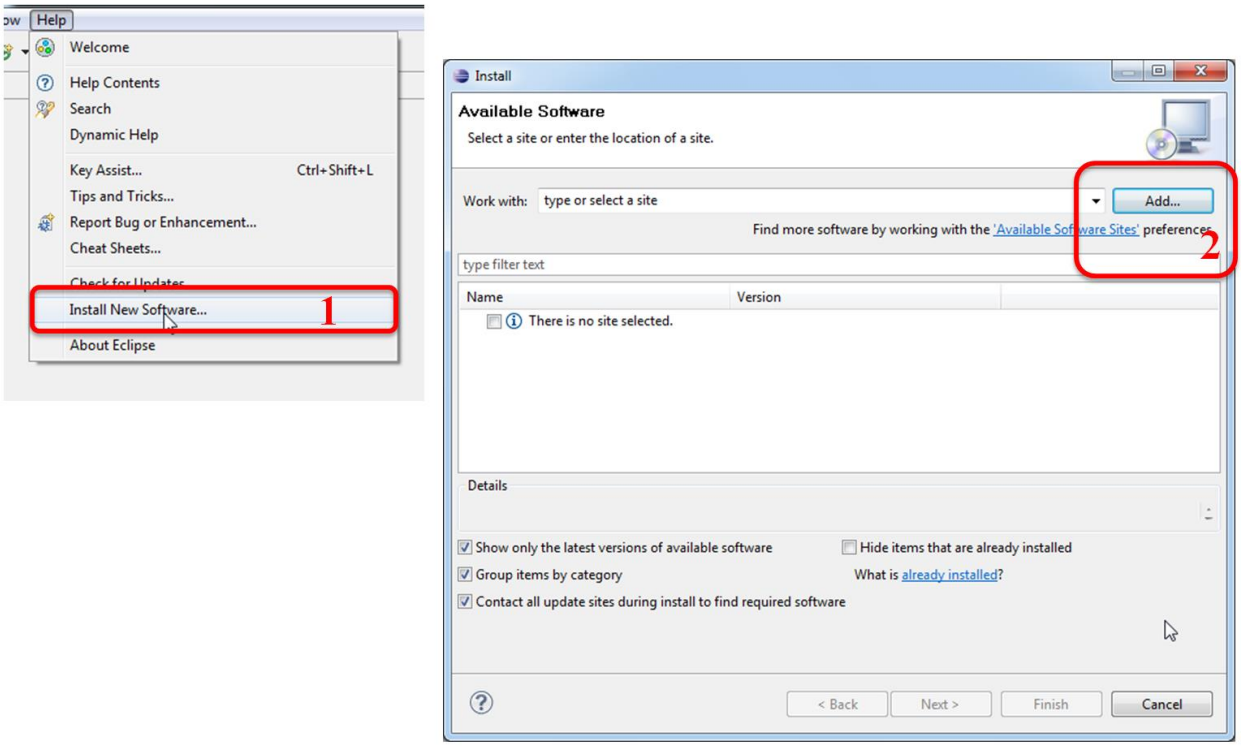
2. How to install Traceino in Eclipse

Traceino comes with a set of plugins for the Eclipse platform, e.g. for creating case specific traceability meta models with the aid of standard Ecore model editors and additional views provided by Traceino for modeling traceability specific aspects like the traceability link type description and constraints for the trace references of a traceability link type. In addition, Traceino provides basic implementations for Eclipse based editors in general and GMF based editors in particular which heavily eases the implementation of a Traceino adapter for an Eclipse based tool. Another aspect of the Eclipse distribution of Traceino is its traceability querying facility which is still in an experimental state and not fully implemented yet. All of these features and adapters for some Eclipse based tools like the Requirements Engineering Platform ProR (see http://www.eclipse.org/rmf/pror/) or the Papyrus UML Tool (see http://www.eclipse.org/papyrus/) can be installed in any tool based on Eclipse Indigo or a newer version of Eclipse.

It is suggested to use the pre-bundled Eclipse Modelling Tools package since it already includes a number of tools needed for Traceino. You may use the following link for downloading an Eclipse Modeling distribution:

http://www.eclipse.org/downloads/

You can install Eclipse by just unpacking the archive to the location you prefer. Then start Eclipse and call “Install New Software…” from the “Help” menu and press “Add…” in the window popping up (see Figure 1).



1. **Figure 1 Adding New Update Site**

In the window appearing type the location of the update site – you find it in the download section of Traceino on the ModelBus® website (see http://www.modelbus.org/en/traceinodocumentation.html). You should also state a name for the update site, e.g. “Traceino” (see Figure 2).

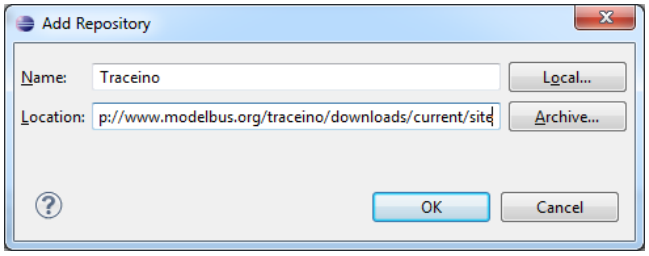


Figure 2 Add a New Update Site

After pressing OK, the software available within the update site will be displayed in the Install dialog. The Traceino update site contains three categories (see Figure 3). Within the *Traceino Core* category, you will find the tool integration feature, the feature for modeling case specific traceability meta models and the experimental traceability query feature. You may select at least all features from this category except the latter. The *extension* category contains some extensions for Traceino, e.g. for modeling constraints (Epsilon EVL and Topcased OCL feature), for the graph representations of traceability links (Eclipse Zest

feature) and – most important – for a specific traceability provider. Traceino is shipped with a standard implementation based on our powerful tool integration framework ModelBus® (see http://www.modelbus.org/modelbus). You may choose all of these features to be installed. If you want to test the current (incomplete) implementation of the traceability query facility based on ModelBus® and EMF Query2, you may also select the respective features contained in this category. The *Tool Adapters* category provides Traceino adapters for a set of Eclipse based tools. You can also install them if you already have the corresponding tools installed within your Eclipse installation.

Start the installation by pressing "Next" (see Figure 3). You will be guided through the next steps of the installation by Eclipse. After having installed the desired Traceino features, you are required to restart Eclipse in order to apply the changes properly. Then you are ready to use Traceino and model a specific traceability meta model for instance as described in section 3.1.1.

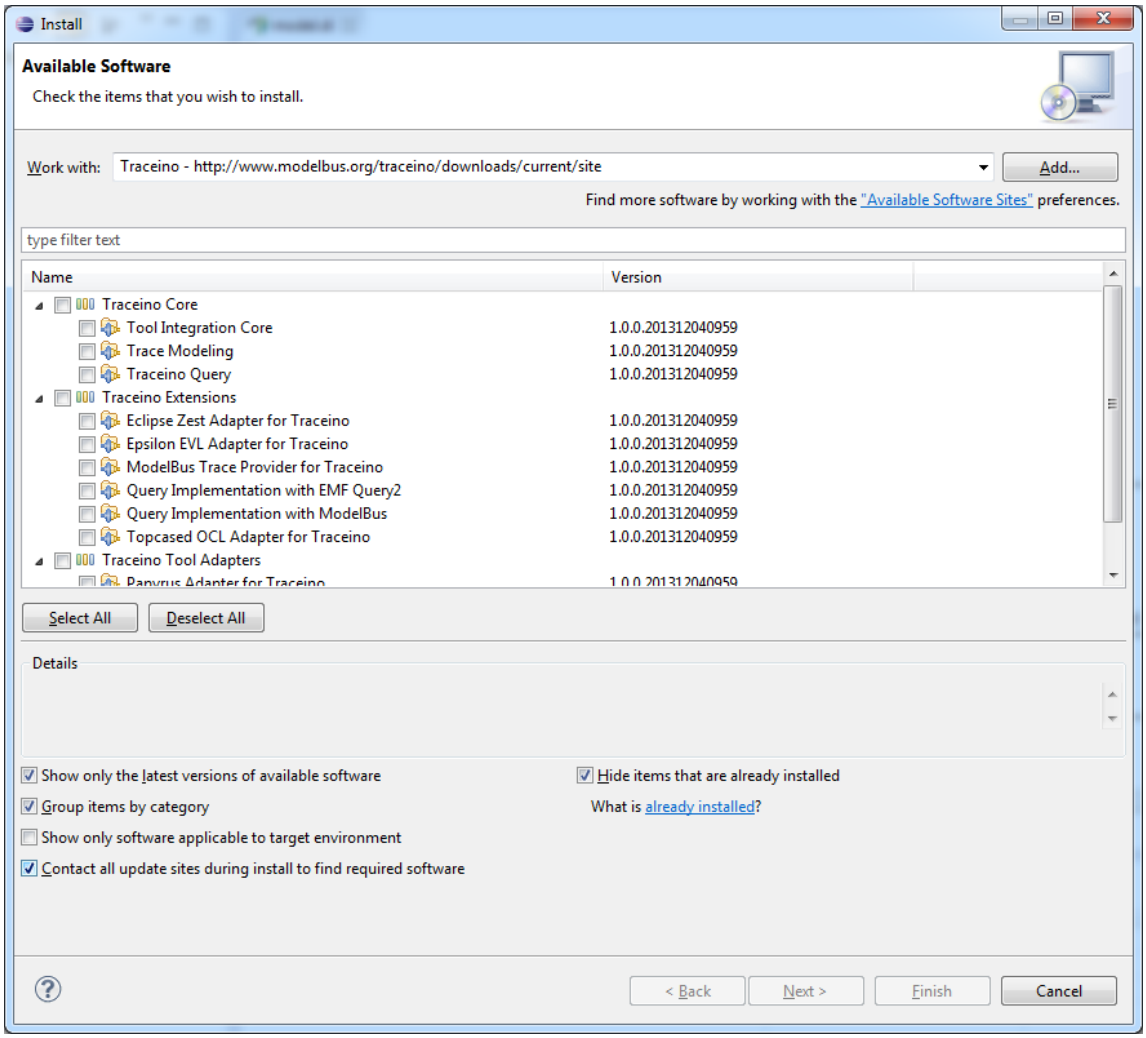


Figure 3 Available Software in Traceino Update Site

3. How to use Traceino

The Traceino framework mainly consists of two parts. One concerns about creating case specific traceability meta models with type safe traceability link types, while the other deals with exploring and managing traceability links in the tools the to be linked elements themselves are managed in. This section explains how to define a custom, case specific traceability meta model using the framework and how to manage traceability links with the help of the user interfaces the Traceino framework offers.

3.1 Traceability Meta Model

Traceino is built on top of the Eclipse Modeling Framework (EMF). Therefore, it uses standard EMF editors like the tree editor or the diagram editor to model case specific traceability meta models. These editors are supplemented by a special view provided by the Traceino framework which allows capturing traceability specific aspects. This section describes how to create traceability meta models within Traceino and how to share them in a common repository with the help of a traceability provider based on ModelBus® technology which is included in the Traceino distribution.

3.1.1 Creating a Traceability Meta Model

Creating a traceability meta model requires to create an EMF project and an Ecore model in that project first. In order to create a new EMF project, please select *File > New > Other…* in the Eclipse application menu. In the “New” wizard, please choose the option “Empty EMF Project” within the “Eclipse Modeling Framework” section and click “Next >” (see Figure 4).

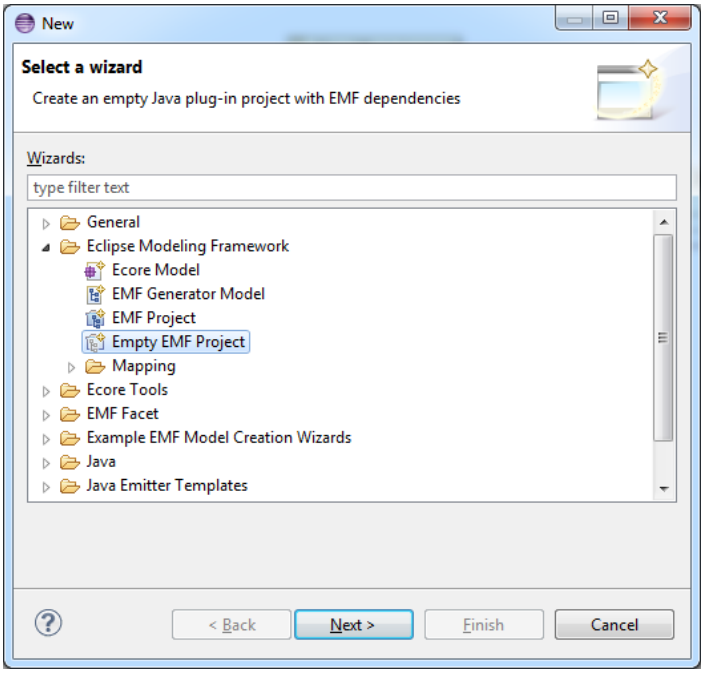


Figure 4 Wizard for Creating a New EMF Project

The next wizard page asks you to specify a name for the project to be created. Please choose a project name of your choice, e.g. *org.tracescenario.model*, and create the project by clicking the "Finish" button. Within the Project Explorer, please expand the entry of the newly created project and select the "model" folder (see Figure 5).

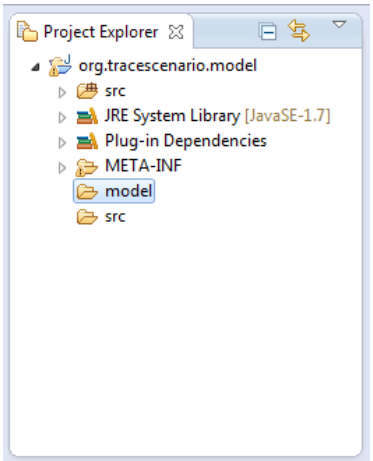


Figure 5 Structure of the EMF Project

Open the context menu of the "model" folder and select New > Other… In the appearing "New" Wizard, please select the option "Ecore Model" within the "Eclipse Modeling Framework" section and click "Next >" (see Figure 6).

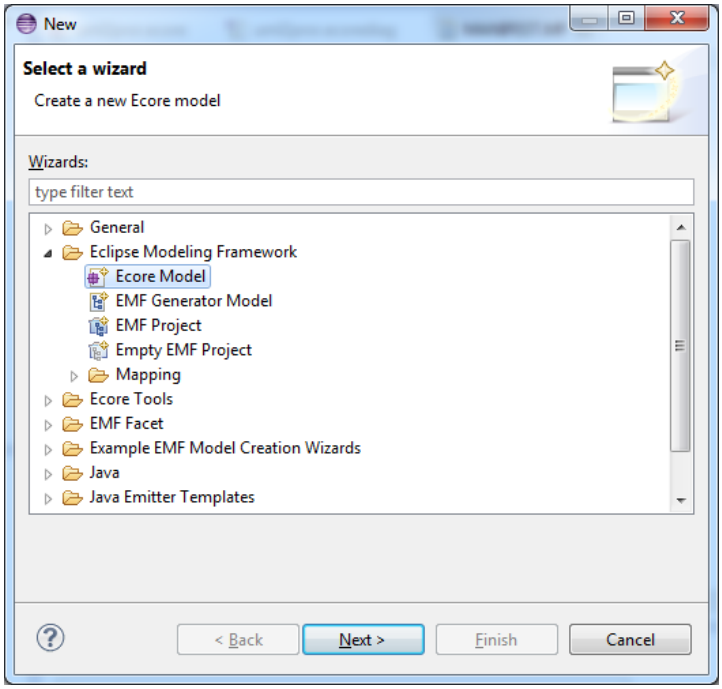


Figure 6 Wizard for Creating a New Core Model

In the next step, please choose a name for the model, e.g. *tracing.ecore*, and click the "Finish" button to create the Ecore model. After the model has been added to your project, Eclipse opens the Ecore Model Editor which allows you to design your traceability meta model using a tree-like representation of your model. Within the tree, please select the unnamed package which has been created for default, open its context menu and select the option "Show Properties View". This view allows you to edit the structural properties of the model element in context. In this case, you should give the package a name and a prefix (*Ns Prefix*) for its namespace of your choice. In addition, you are required to specify an URI (*Ns URI*) for its namespace. In the context of the example used in this guide, this could be something like shown in Figure 7.

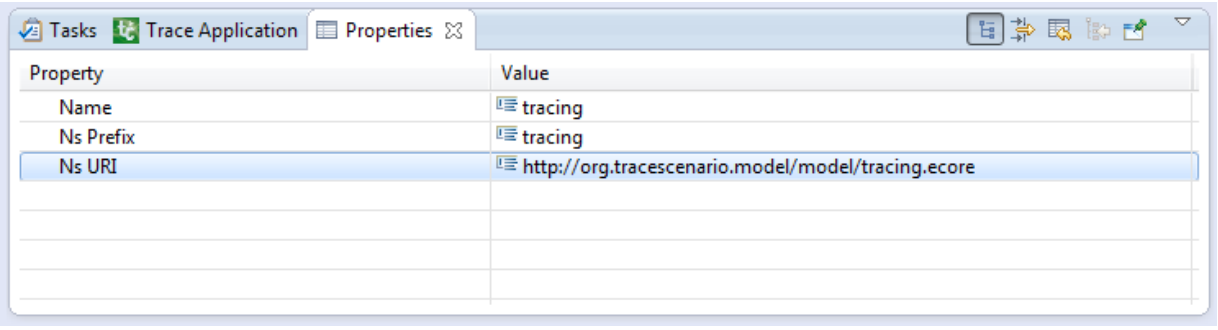


Figure 7 Properties of the Root Package of the Traceability Meta Model

Please note: It is recommended to derive the URI of the meta model's namespace from the location of the model file within the project. In the example of the guide, the model file is contained in a folder called "model" which itself resides in the project "org.tracescenario.model". This path is used for the namespace prefix in conjunction with the HTTP protocol.

After having specified the package's properties, you can start to model your traceability meta model as required in your specific use case. Traceability link types are modeled as *EClasses* which means that they can own a set of custom or case specific properties. Moreover, traceability links created with Traceino are type safe. You are required to add references to the classes of the model elements that can be linked by an instance of the traceability link type. The references are modeled by *EReferences* to *EClasses* that belong to different meta models.

In the following, the way traceability link types are modeled in Traceino is explained by using a simple link type that links requirements modeled within the Eclipse based tool *ProR* (see http://www.eclipse.org/rmf/pror/) with UML classes modeled using *Papyrus* (see http://www.eclipse.org/papyrus/) which is an Eclipse based tool, too. You are required to install both tools before proceeding with the tutorial.

In order to create the above mentioned traceability link type, we will create an EClass *Class2Requirement* first by selecting *New Child > EClass* in the context menu of the *tracing* package.

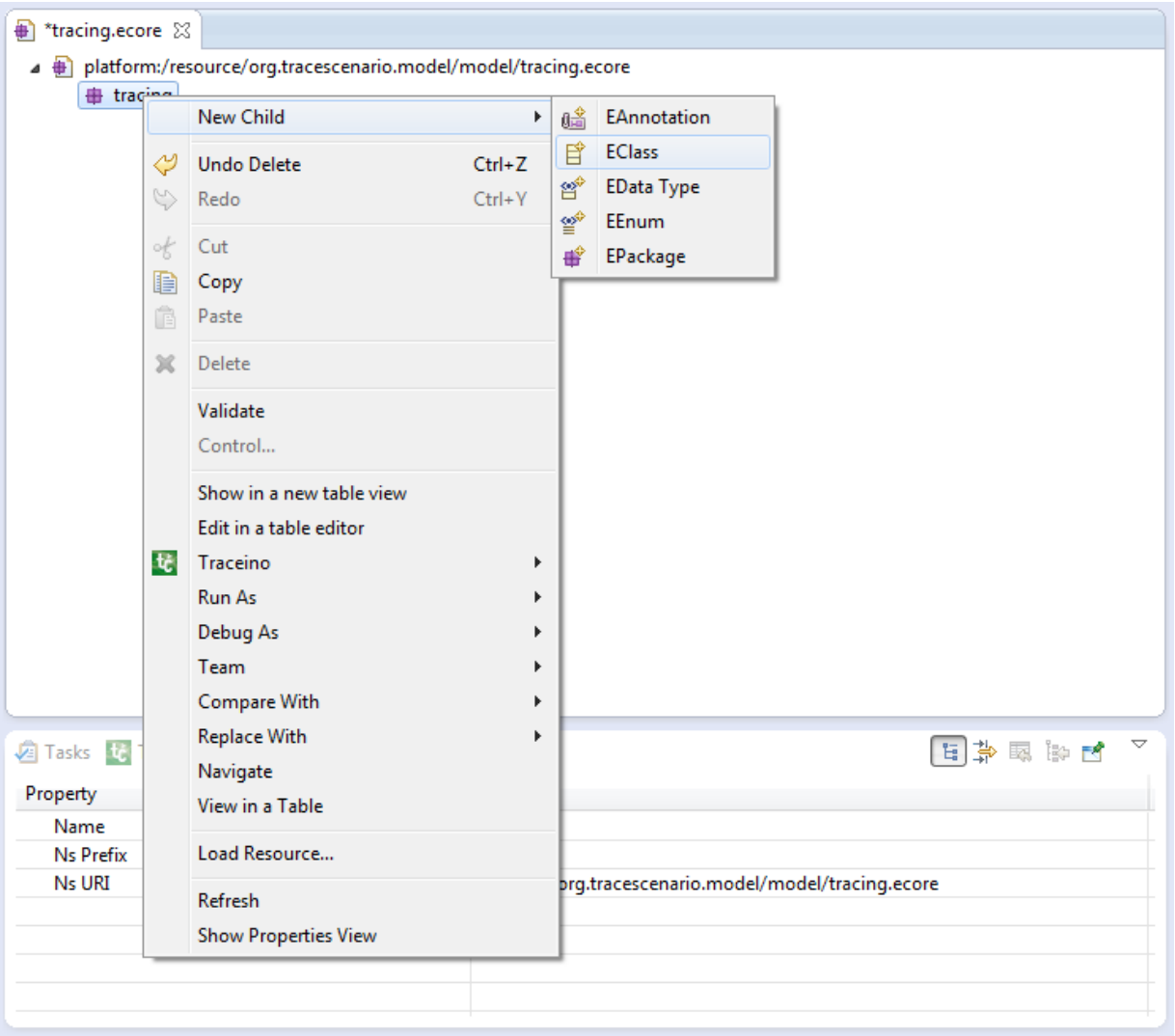


Figure 8 Creating a New Traceability Link Type

This adds a new and unnamed *EClass* to the package which should be named *Class2Requirement* using the Properties view as explained in the context of the creation of the *tracing* package.

The next step is to model the trace link type's references to the *EClass* which is used to model requirements in *ProR* and to the *EClass* which is used to model classes in *Papyrus*, respectively. In order to get access to these types, you are required to "import" the meta models of these tools into the EMF editor first. Therefore, please open the context menu of the EMF editor showing your traceability meta model and select the option "Load Resource…" (see Figure 9).

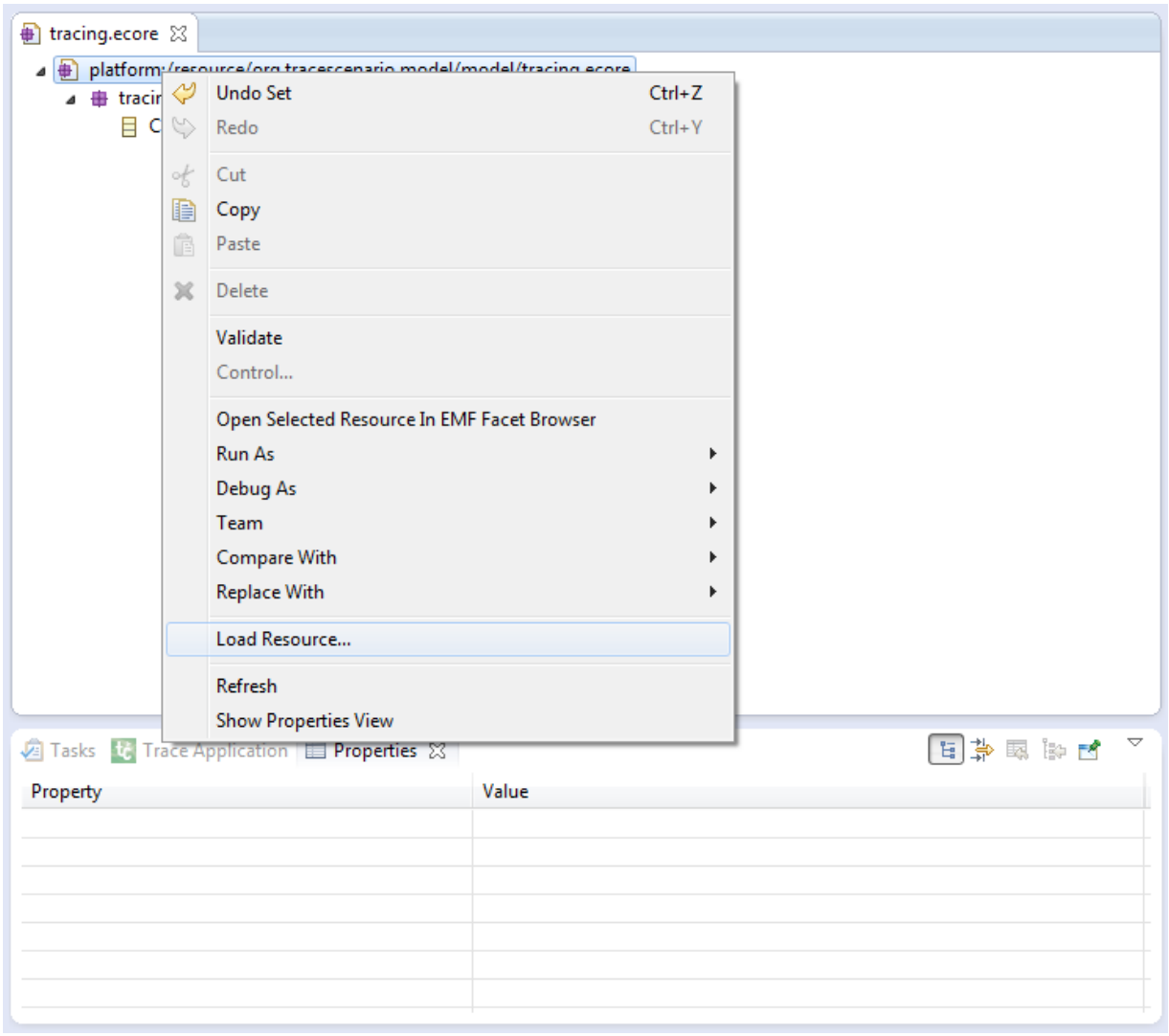


Figure 9 Loading Additional Meta Models into the EMF Editor

In the dialog popping up, please click on “Browse Registered Packages” to load a meta model already installed in the Eclipse workbench (see Figure 10).

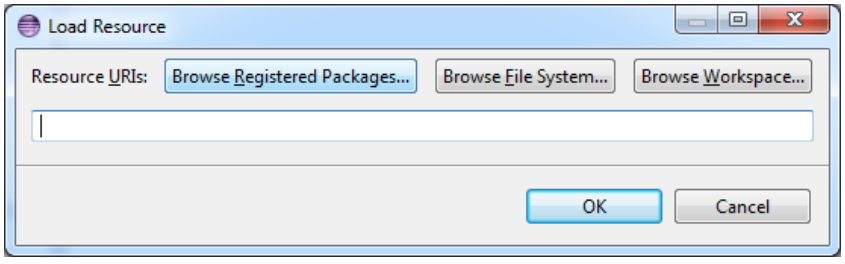


Figure 10 Loading an Installed Meta Model into the EMF Editor

This opens a dialog which allows you to select registered packages to be loaded into the editor. Please select the packages http://www.eclipse.org/uml2/4.0.0/UML (or a different

version) and http://www.omg.org/spec/ReqIF/20110401/reqif.xsd to load the classes needed to model the traceability link type (see Figure 11).

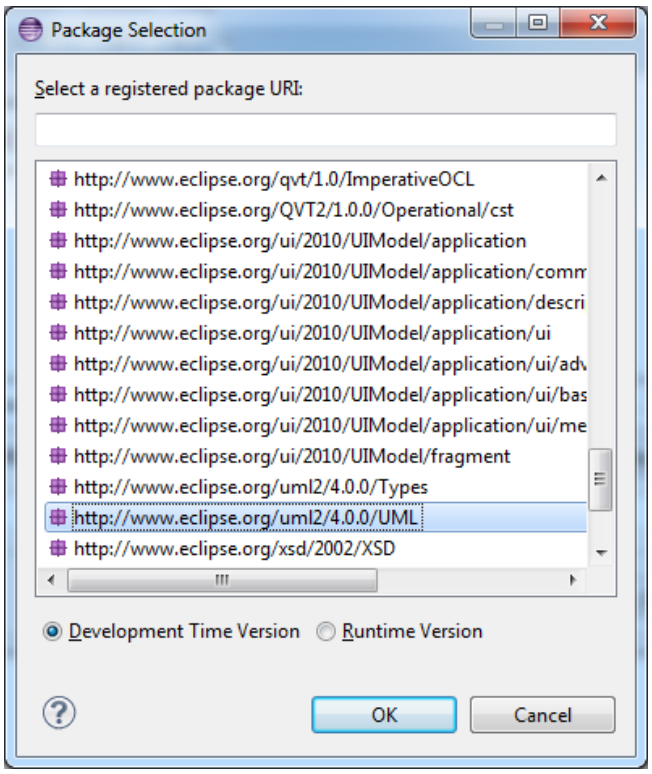


Figure 11 Loading the required UML meta model package into the EMF Editor

Please note: If a package is not included in the package URI list, you should switch to meta models available at runtime only by clicking the corresponding radio button below the list.

After having loaded the required packages into the editor successfully, you are able to create the references of the traceability link type. Therefore, open the context menu of the *Class2Requirement* class created beforehand and select *New Child > EReference*. You should name the reference according to its role, e.g. “requirement” or “class” for the reference to link requirements or classes, respectively.

In addition, you are required to set the type of the reference according to your needs. In this tutorial, we will use the EClass *SpecObject* of the ReqIF meta model for the *requirement* reference since this class is used to model requirements in ReqIF or ProR, respectively. In case of the class reference we choose the EClass *Class* since it represents classes in UML or Papyrus, respectively.

Furthermore, you should specify the minimum number (*Lower Bound*) and the maximum number (*Upper Bound*) of elements that can be linked by this trace link type reference. In this tutorial, we do not restrict the maximum number of classes or requirements to be linked by an instance of the *Class2Requirement* traceability link type and therefore select "-1" as the upper bound value for both references.

Please use the singular for naming the trace link type references even in the case of allowing the linkage of multiple elements since Traceino will automatically derive the plural form when required in a particular context.

If you prefer modeling an Ecore model using the Ecore Diagram Editor, you can use this editor since it is also integrated with Traceino.

Up to now, we have modeled a single EClass owning two EReferences which should represent the traceability link type. The next step is to tell the Traceino framework, that both EReferences should be used as trace references and thus can be extended by additional traceability specific aspects. To do so, you are required to open the context menu of both references and select *Traceino > Use As Trace Reference* (see Figure 12).

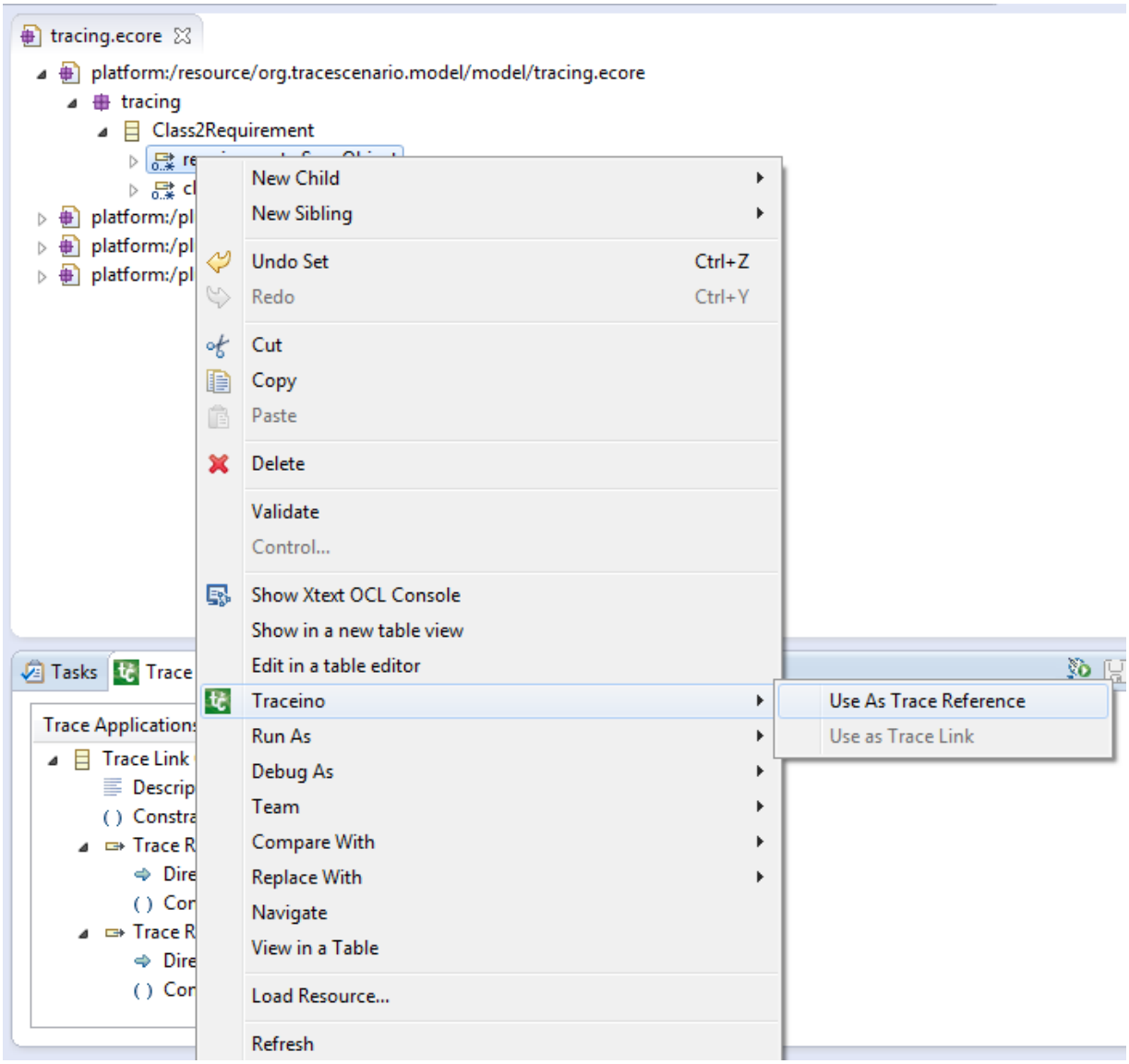


Figure 12 Marking a Reference As Trace Reference

This will create an additional model in the model folder which is able to capture additional traceability specific aspects like the rationale of a trace link type or constraints for the trace references expressed in languages like OCL or EVL. This model is named according to your Ecore model, but owns the suffix “traceapplication”. You do not have to pay attention to this model since it is not of particular interest for a user. But please do not modify it manually or even delete it, since it is later on required to create the final traceability meta model.

The additional traceability specific properties mentioned above can be modified using a special view which is called “Trace Application”. If you do not have this view included in your perspective, you should add it by choosing *Window > Show View > Other…* from the

application menu. In the dialog appearing next, please select the corresponding view from the Traceino section (see Figure 13).

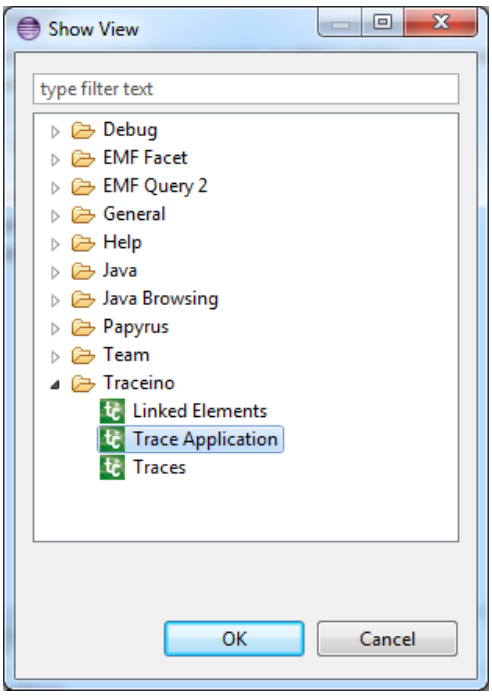


Figure 13 Enable Trace Application View in Eclipse

For example, the Trace Application view allows you to specify a rationale for a trace link *type*. Therefore, please select the description item of the trace link type and enter a descriptive text in the text field appearing at the right side of the dialog (see Figure 14). In the context of the *Class2Requirement* trace link type, you can state something like “This trace link type links requirements modeled in ProR with UML classes modeling in Papyrus.” or something comparable.

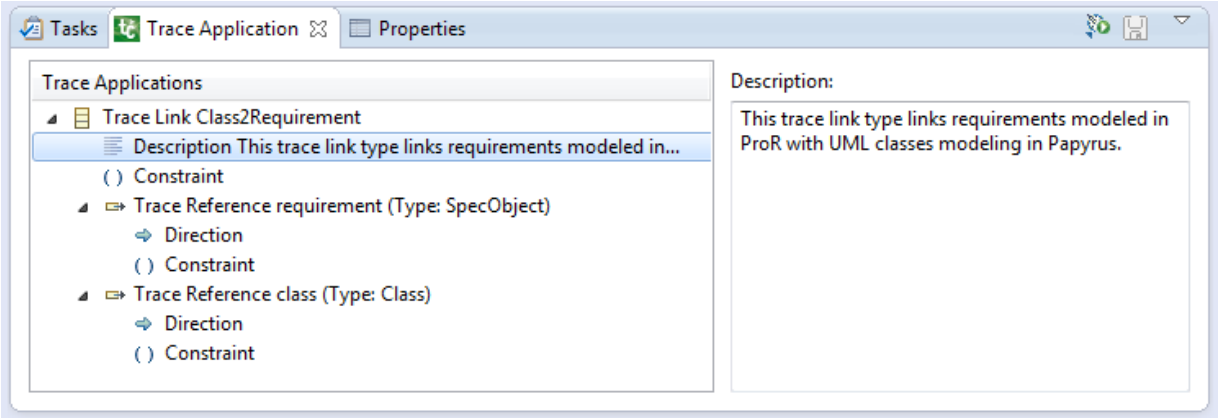


Figure 14 Specifying the Rationale of a Trace Link Type

Beside the rationale of the trace link type, you should specify the directions of the trace link references within the processes. A very important aspect of a trace link is its direction, for example the information weather it links model elements from an abstract level to a more concrete one (down) or vice versa (up). A trace link type created with Traceino can own multiple trace references which allows linking elements in different directions within the same traceability link. Therefore, the direction of the linkage between different sets of elements should be expressed in the context of a trace reference. The Trace Application dialog allows capturing the direction of each trace reference of a particular trace link type. In order to specify the direction for a given trace reference, please select the “Direction” item of the corresponding trace reference. This will display a form at the right side of the dialog which allows you to select an appropriate direction or create a new one (see Figure 15).

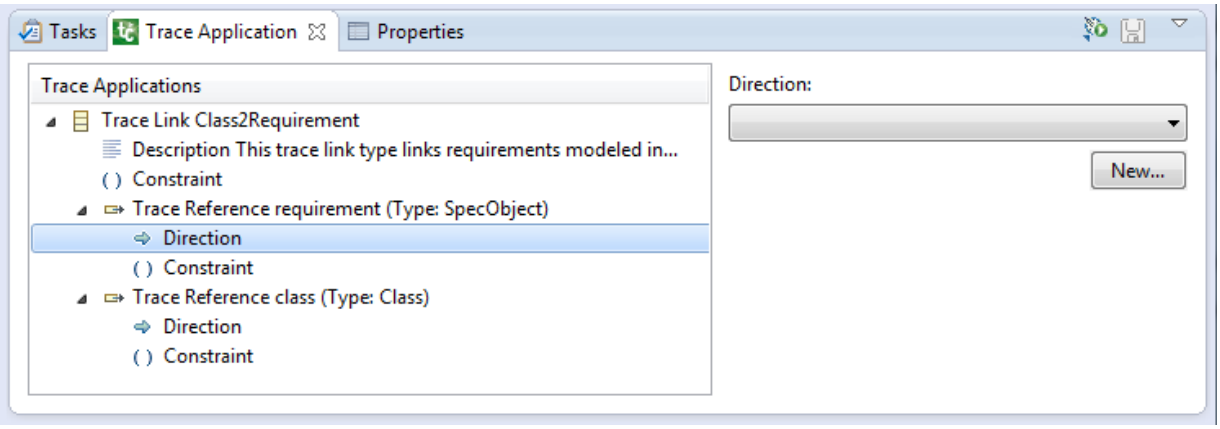


Figure 15 Specifying the Direction of a Trace Reference

You can either select one of the two basic directions “top” and “down” representing both “general” vertical directions as described above. Alternatively, you can create new directions and derive them from existing ones to create a more sophisticated direction hierarchy which eases the development and increases the potential of analysis processes examining the traceability information. In our example, we will create a new direction “toRequirement” for the *requirement* trace reference and derive it from the *up* direction. Furthermore, we will create the direction “toDesign” derived from the *down* direction and assign it to the *class* trace reference. The result looks like shown in Figure 16.

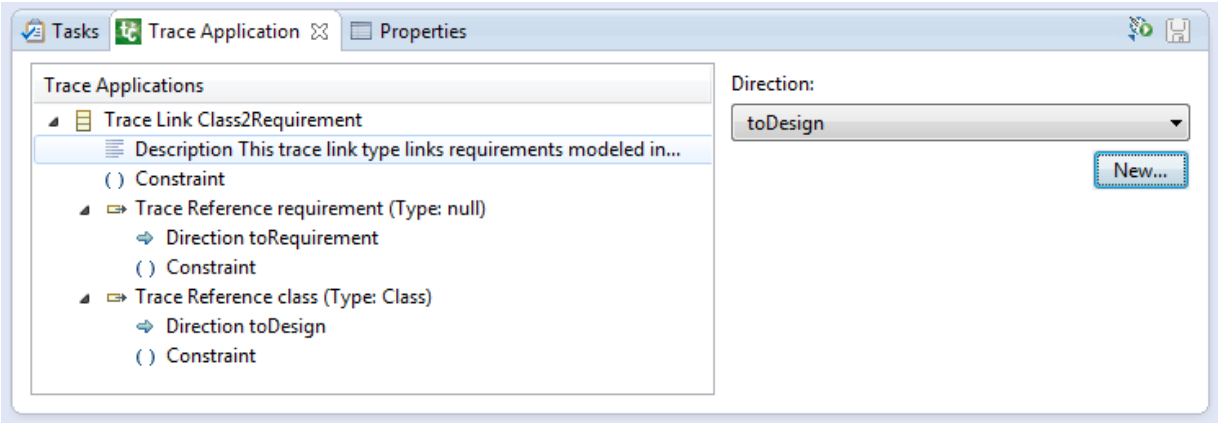


Figure 16 Directions for the Trace References used in the Tutorial

Finally, you can specify some constraints for the trace references using wide spread constraint languages like OCL and therefore restrict the set of elements that can be potentially linked by an instance of the corresponding traceability link type. When clicking of the "Constraint" entry of a trace reference, the Trace Application view offers a form that allows defining a constraint for the given trace reference (see Figure 17).

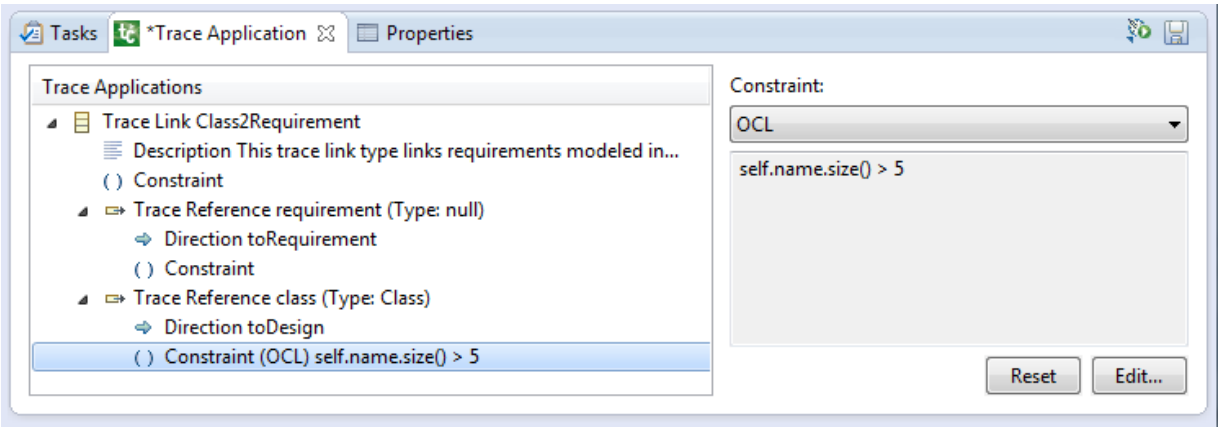


Figure 17 Specifying a Constraint for a Trace Reference

Figure 17 shows a (not necessarily meaningful) constraint for the *class* trace reference of the *Class2Requirement* trace link type modeling in this tutorial. The constraints is written in OCL and states that a class must have a name longer than five characters to be able to be linked by an instance of the traceability link type. If you want to edit a constraint, you have to click on the "Edit…" button below the read-only constraint text field. This will open up the editor installed for the given constraint language that allows you to modify the constraint while using the editor's specific support features like code completion and syntax highlighting. Figure 18 shows the Topcased OCL editor for the constraint of the *class* trace reference.

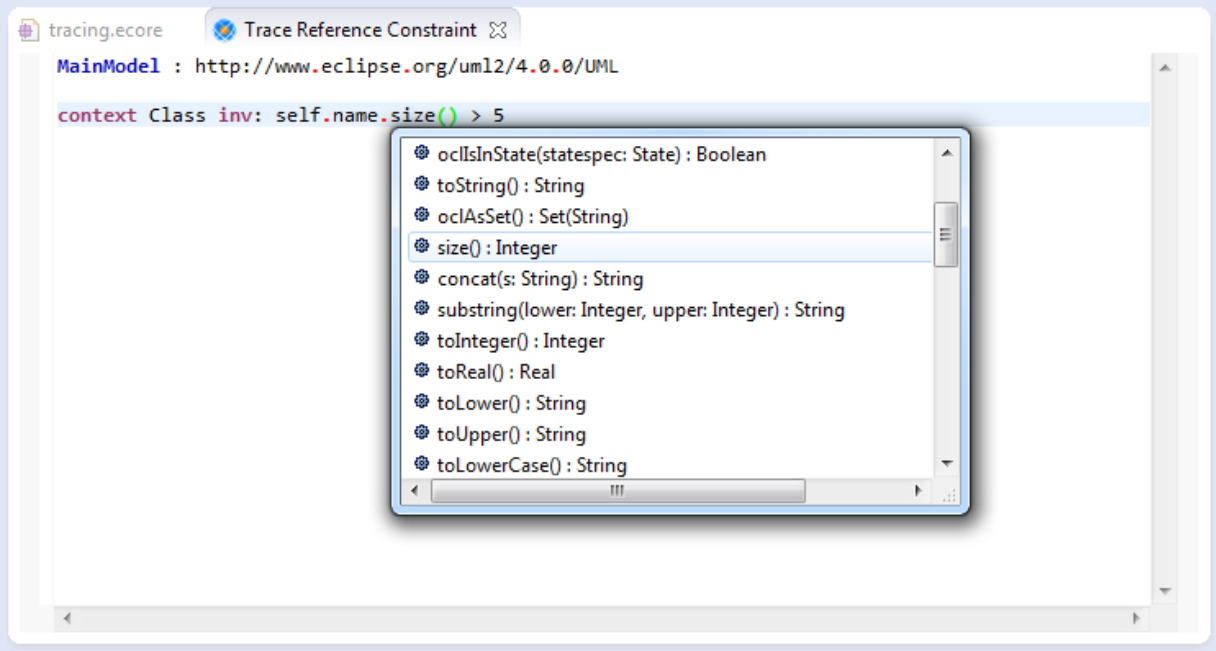


Figure 18 Topcased OCL Editor Showing a OCL Constraint

After having saved the changes made in the constraint editor, they will be reflected into the Trace Application model automatically.

Please note: In order to be able to specify trace reference constraints, you must at least have one Traceino adapter for a constraint engine like installed into Eclipse. Traceino is already shipped with adapters for Topcased OCL and Epsilon EVL as described in section 2, but also extensible by adapters for different constraint engines.

When the design of the traceability meta model is completed, i.e. after having modeled the structural features with the aid of the standard EMF editors and the additional traceability specific aspects with the help of the Trace Application view, you are required to combine both models, the Ecore Model and the Trace Application model, into a single traceability meta model. The latter is also called *Tcore model* which represents a "format" the Traceino tooling can deal with. This is a fully automated step. In order to invoke the transformation, please click on the "Transform to TCore" button in the upper right corner of the Trace Application view (see Figure 19).

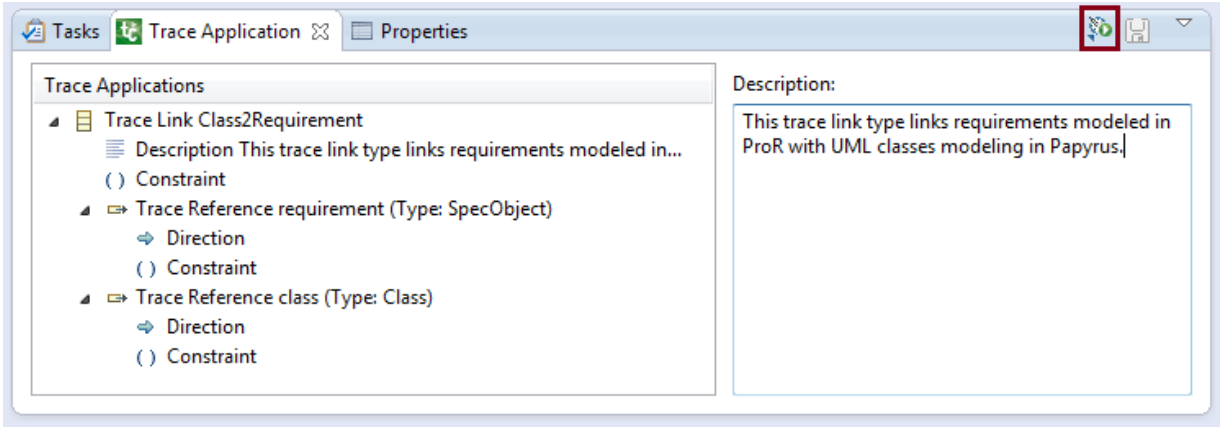


Figure 19 Invoking the Automated Creation of the Tcore Model

This will create a third model in the model folder called “tracing.tcore.ecore” which is used in the traceability framework (see Figure 20). The reason for this additional transformation step is the fact, that EMF is restricted to a three layered architecture and therefore does not support the “instantiation” of the case specific traceability meta model. These trace models would reside on a forth model layer which is not possible in context of EMF.

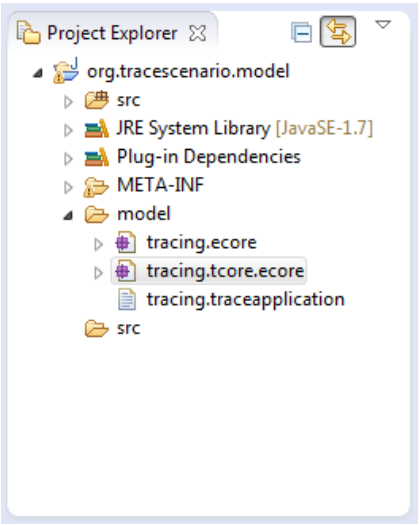


Figure 20 Generated Tcore Model in the Project Explorer

Now, the creation of the case specific meta model is completed and the meta model is ready to be used with Traceino. The following section describes how a traceability meta model can be committed to a ModelBus Repository opening the potential to be used in a distributed scenario and in different tools integrated with Traceino.

3.1.2 ModelBus® Provider: Sharing a Traceability Meta Model in ModelBus® Repository

Traceino can be extended by so called “Trace Providers” which offer access to the model elements that can be linked in a given project or process and the traceability links between these elements. A trace provider also offers the opportunity to create new traceability links or to manage existing ones.

Traceino is shipped with a default Trace Provider based on the ModelBus® technology which is also developed at Fraunhofer FOKUS (see http://www.modelbus.org/modelbus for more information). If you have the ModelBus® Team Provider and the corresponding trace provider for Traceino installed into Eclipse (see section 2), you can share models like domain models and traceability models and other artefacts in a common ModelBus® repository that offers great features like the versioning of artefacts, the partial loading and merging of model artefacts and an in-depth model dependencies management. The latter enables Traceino to efficiently “discover” models of a particular meta model within the repository and traceability links between those models and to load these models together with the whole set of models they possibly depend on.

 Please see the ModelBus® users guide (http://www.modelbus.org/modelbus/index.php/documentation/users-guide) for details regarding the **installation** process and the **configuration** of the ModelBus® Server and Team Provider.

Before trying to share a traceability meta model in a ModelBus® repository, please make sure that the dependency management feature is enabled for the file extensions of the models in charge. Therefore, please select *Window > Preferences* from the application menu of Eclipse and select the “ModelBus” section in the preferences dialog. Within that section, please choose the “Dependencies” entry (see Figure 21). In the Dependencies dialog, please check “Enable dependencies analysis” and ensure, that all file extensions of the models in scope are added to the list of managed models.

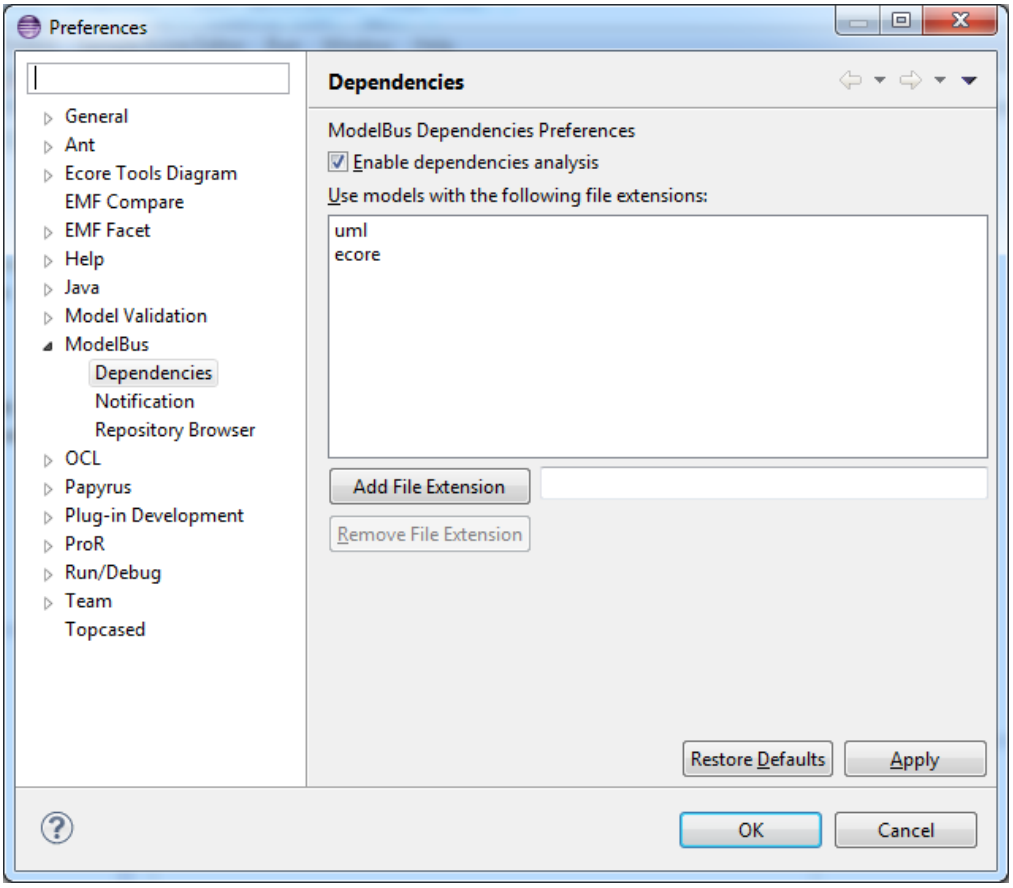


Figure 21 Enabling Dependencies Management in ModelBus®

For further information related to the ModelBus® dependency management feature please see the ModelBus® users guide (http://www.modelbus.org/modelbus/index.php/documentation/users-guide).

When the ModelBus® dependency management is setup correctly, you can commit the traceability meta model (Tcore model) and its related meta models into the repository. In order to do that, please select the project that contains the Tcore model in the Project Explorer, e.g. *org.tracescenario.model*, and open the project's context menu. In this menu, please select the option *Team > Share Project…* (see Figure 22).

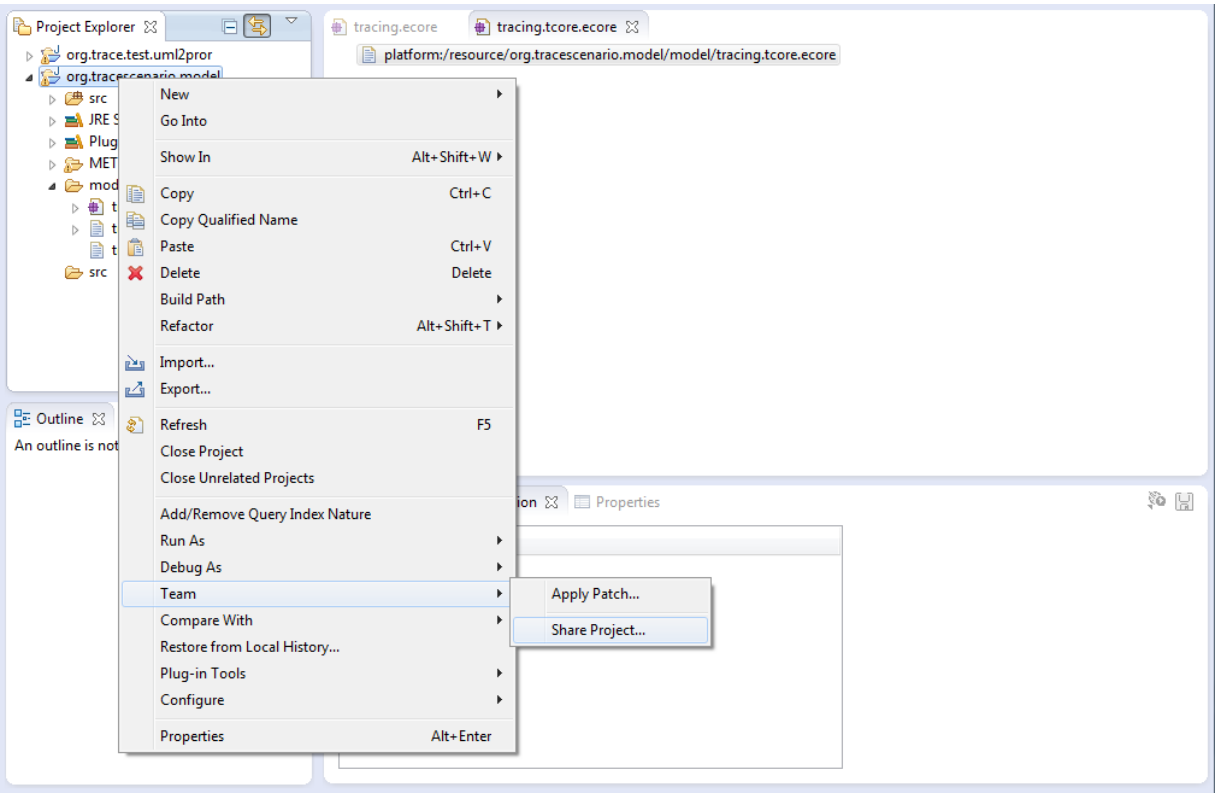


Figure 22 Sharing Traceability Meta Model in ModelBus® Repository (1)

This opens a dialog which asks for the location to be used for sharing the project (see Figure 23).

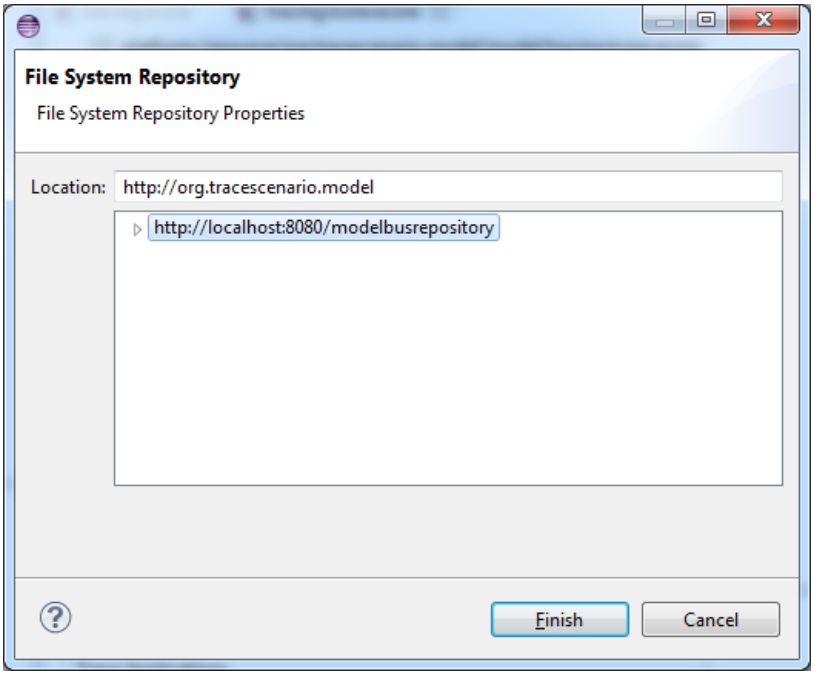


Figure 23 Sharing Traceability Meta Model in ModelBus® Repository (2)

In that dialog, you can simply click on the list entry representing the configured ModelBus® server installation in order to automatically determine the repository location based on the project's name. By clicking on the "Finish" button, the project is shared in the repository, but its content is not committed to the repository yet. In order to achieve the latter, please select the option *Team > Commit…* in the project's context menu (see Figure 24).

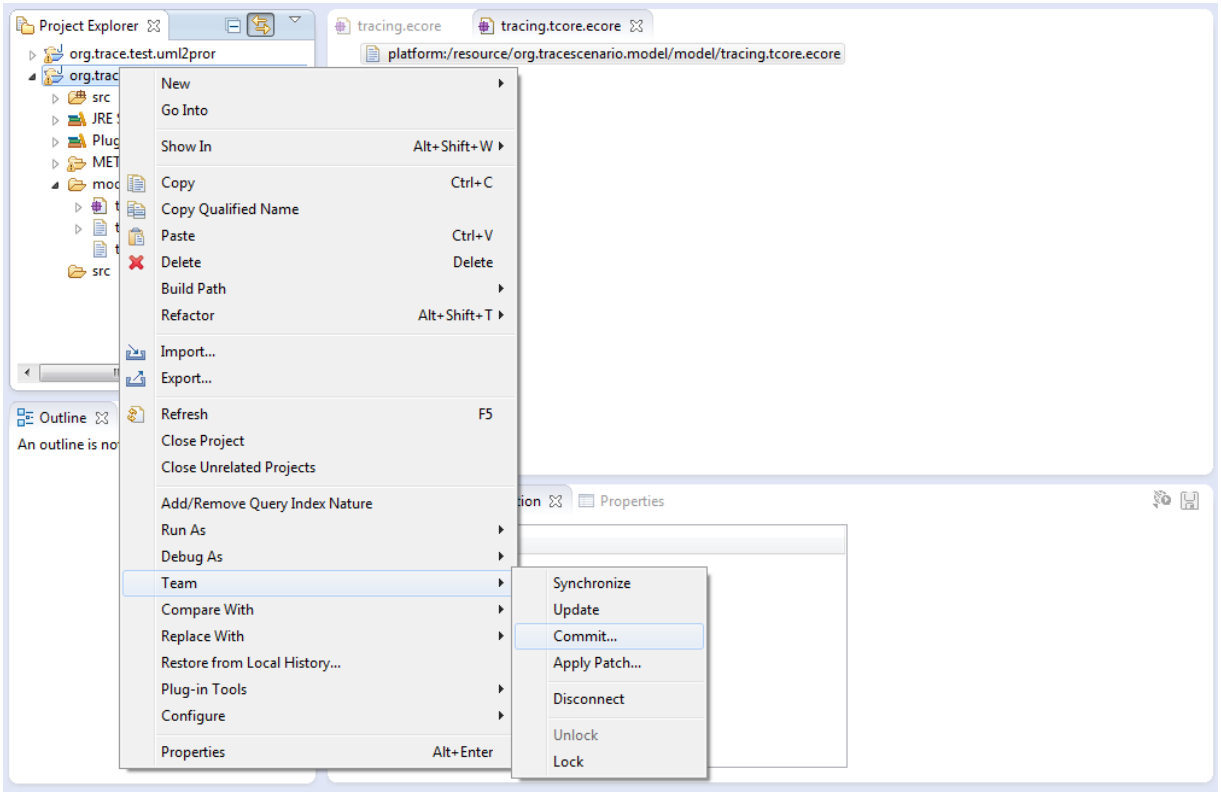


Figure 24 Sharing Traceability Meta Model in ModelBus® Repository (3)

In the dialog appearing, you are asked to enter a commit message. After having done that, the project and its contained traceability meta model will be committed to the repository. This step may take some seconds to be completed, since ModelBus will recursively analyze the dependencies of each model contained in the project and commit them to the repository, too. Once the commit process is finished, the project's name will be followed by an additional information showing the revision number of the project and the repository location the project is shared to (see Figure 25).

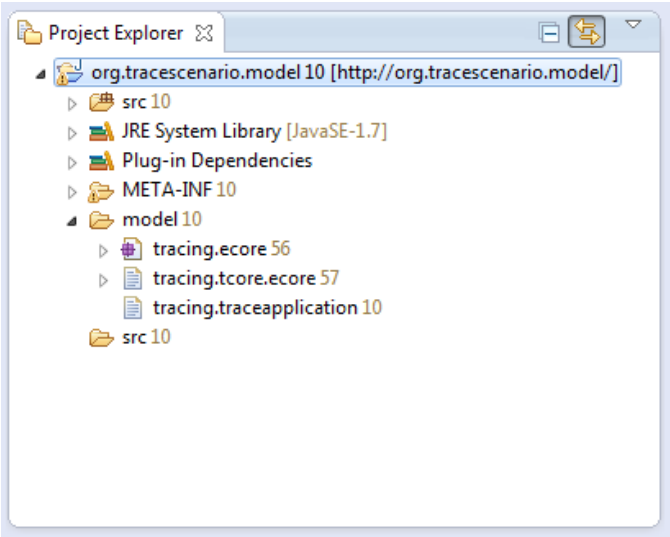


Figure 25 Project Shared in ModelBus® Repository

Now, you are done! The traceability meta model has been successfully committed to the ModelBus® repository and thus is visible to other tools connected to ModelBus®. If these tools are also integrated with Traceino, you can explore traceability links between the model elements of your project and even manage these links or create new ones within the tools themselves (see section 3.2).

3.2 Managing Traceability Links

The main idea of Traceino is to easily extend existing tools with traceability functionality and thereby facilitate end-to-end traceability across tool borders. Therefore, Traceino comes with a set of views and wizards that can be integrated into tools with respect to their user interface concept and capabilities.

This section briefly describes the different views offered by the Traceino framework and their purpose. Moreover, the way how to manage traceability links, i.e. how to view, modify or delete them and how to create new ones is explained.

3.2.1 Traceability Specific Views

The framework is driven by the idea of a traceability approach we call *Explorative Tracing*. This means that a user can explore which model elements a given element selected in a tool is linked to. This is not restricted to model elements managed by the tool in context, but rather open to any model element that is involved in a project. All model elements are visible and, based on the traceability meta models defined in the project, linkable within each involved tool, even if these elements are not originally in the scope of the tool or intended to be managed by it.

The graphical user interface mainly consists of a set of views that can be flexibly integrated into arbitrary tools. It is not specified how these views should be arranged in a tool in detail, but these views are designed to complement a tool's interface with traceability specific views. To ease integration, these views are implemented based on SWT. For Eclipse based tools, special view parts adapting these views are available.

3.2.1.1 Linked Elements View

The *Linked Elements View* follows the metaphor of the Outline View within Eclipse. The view is a structured overview of all model elements that are linked to a given element, i.e. to the element currently selected in the tool the view is integrated in (see Figure 26). Linked elements are grouped by type. In addition, the view shows how often each element listed in the view is linked to the currently selected element. The view is refreshed whenever the selection in the host tool changes. A light weight tool adapter is responsible for reporting the current selection within the tool to the Traceino framework so that the corresponding views can be refreshed accordingly.

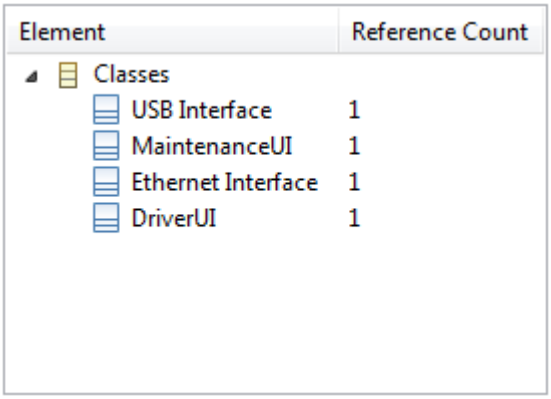


Figure 26 Linked Elements View

3.2.1.2 Traces View

The *Traces View* shows the properties of a given set of traceability links. It is refreshed whenever a linked element in the Linked Elements View is selected. The Traces View usually contains a combo box widget which allows the user to select a single traceability link. The properties of a traceability link are displayed using two different representations – a hierarchical and a graphical representation. The hierarchical representation shows a traceability links as a tree with a number of nodes according to the trace references of the traceability link type that belongs to a given a traceability link. These nodes hold the model elements linked by the trace references (see Figure 27).

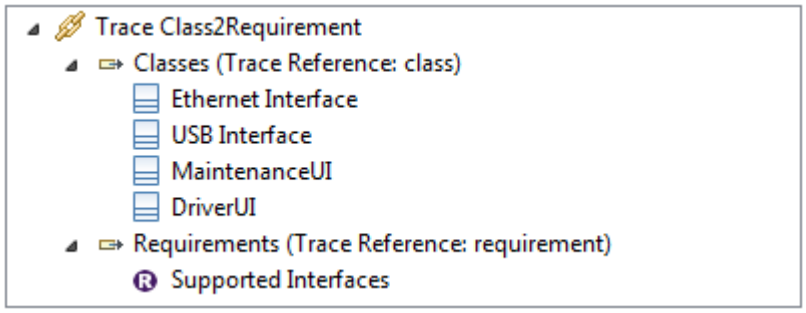


Figure 27 Hierarchical Representation of a Tracability Link

The graph representation displays a traceability link as a graph with a node in the center that represents a traceability link and a number of nodes around this centered node which represent the linked model elements. The way a traceability link is rendered as a graph can be changed by providing a custom extension to the framework. The default implementation which is shipped together with Traceino uses the Eclipse Zest framework (see http://www.eclipse.org/gef/zest/) to render the graphs (see Figure 28). As a result, the nodes of the graphs can be rearranged freely by moving them around with the cursor.

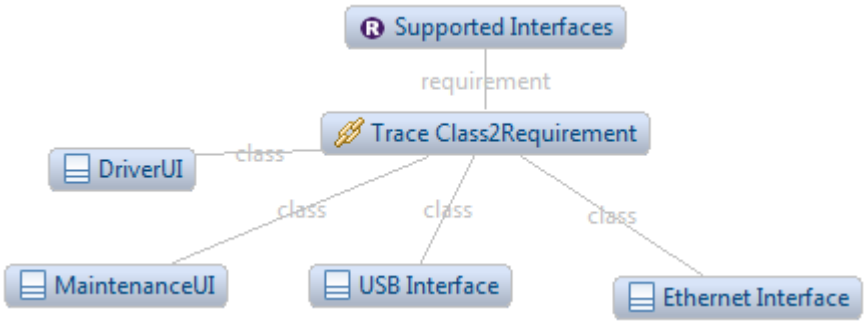


Figure 28 Graph Representation of a Traceability Link

Beside these two representations of a traceability link and its linked model elements, the Traces View displays the rationale of the traceability link and other structural properties that are defined for the type of the traceability link.

3.2.1.3 Trace Properties Wizard

The *Trace Properties Wizard* allows to create new traceability links or to modify existing ones. The wizard covers the following work flow:

1. When creating a new traceability link, the wizard determines with which type of elements the element currently selected in the host tool (context element) is linkable to. Therefore, the framework examines the traceability meta models available. The element types obtained by this analysis are displayed in a dialog so that the user can

choose the type of the elements he wants to link with the context element (see Figure 29).

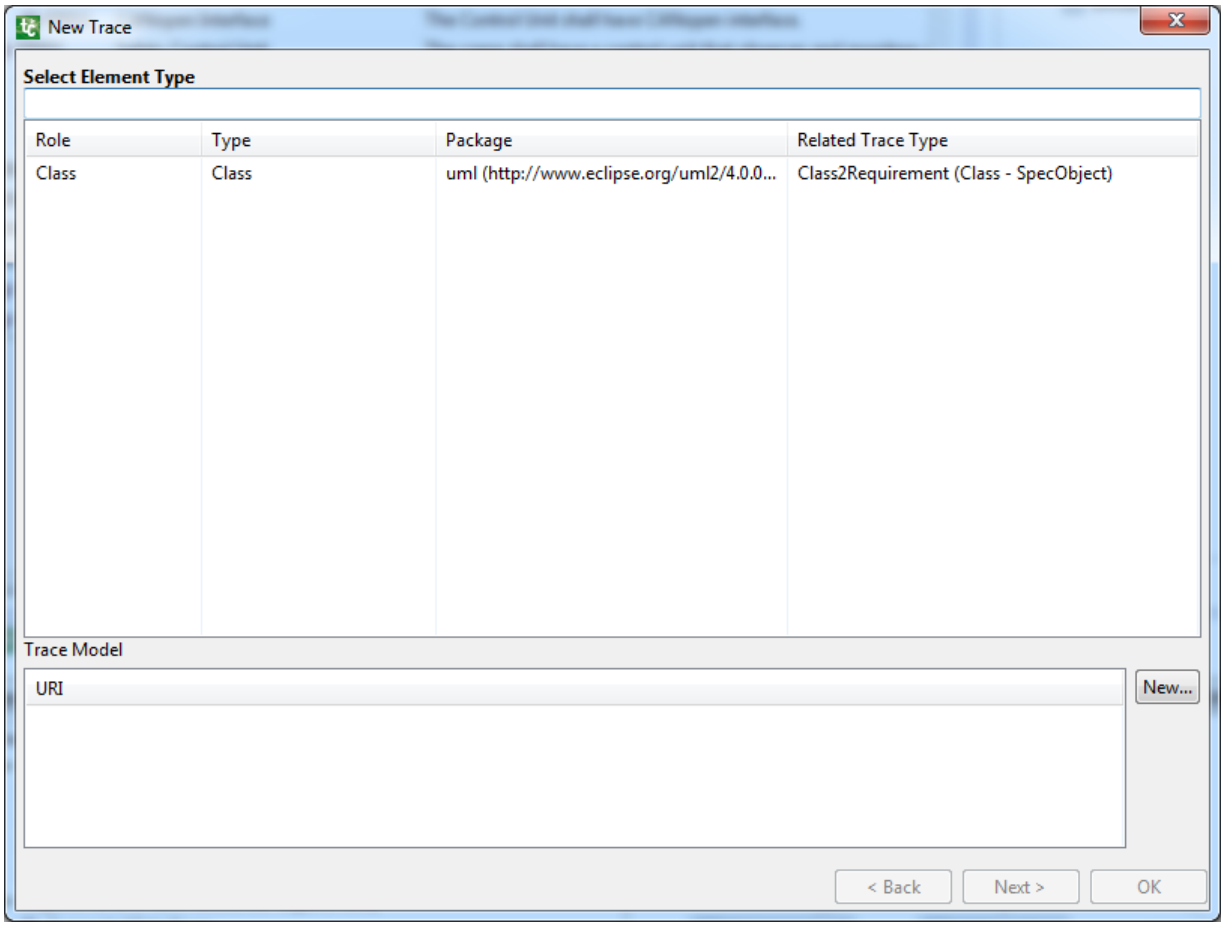


Figure 29 Dialog for Selecting Type of Elements to Link

The selection also implies which traceability link type to instantiate for creating the new traceability link.

2. The wizard offers a set of different filter mechanisms which can help to restrict the potentially huge set of model elements of the type selected in the first step. This includes filtering elements based on:
    a. the location of the model they belong to
    b. the structural features of the elements (e.g. label or other type specific properties)
    c. constraints the user can write to filter elements
    d. specific filters implemented in other languages like Java

    Figure 30 and Figure 31 show the corresponding dialogs for specifying filter conditions.

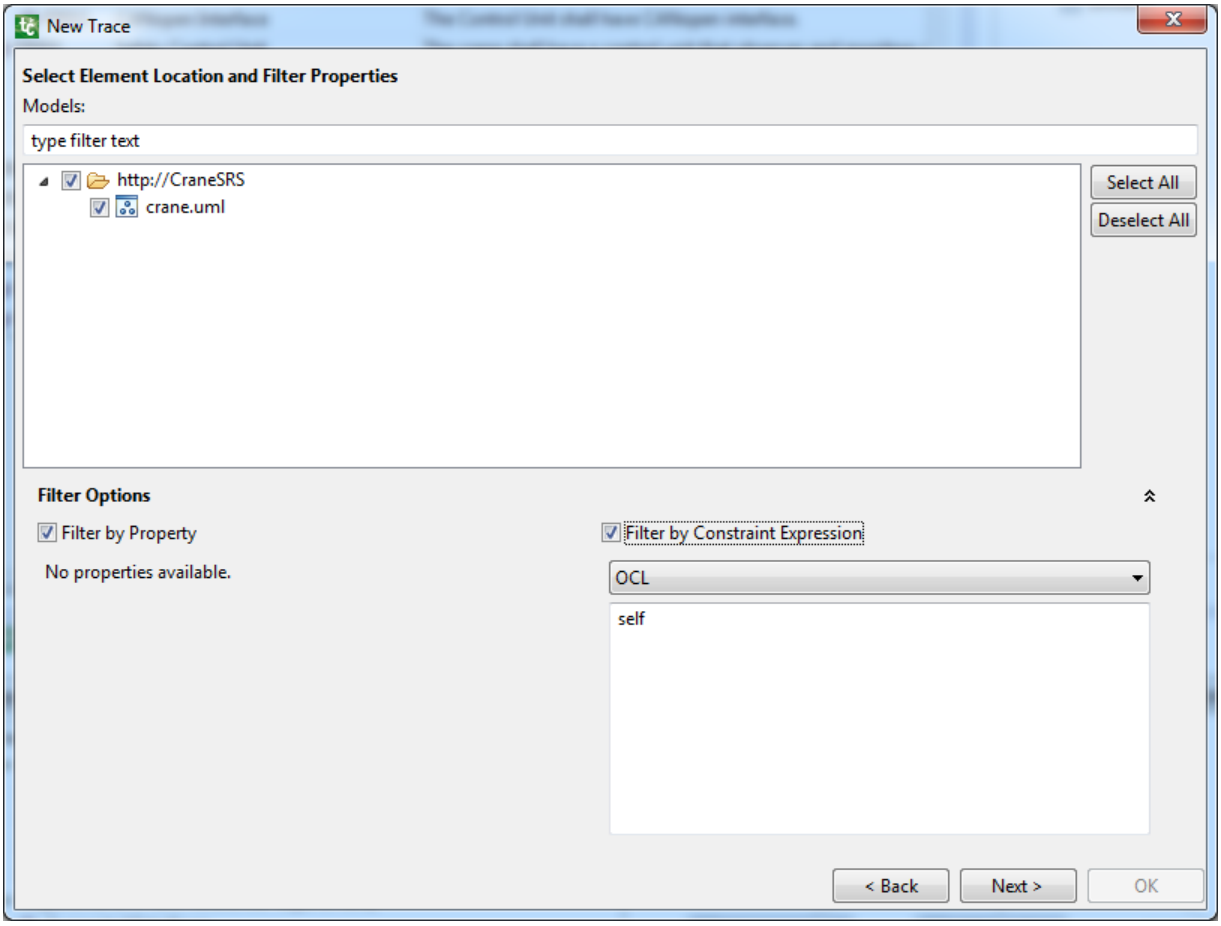


Figure 30 Dialog for Specifiy Element Filters

3. Based on the filter conditions the user has defined in step 2, the wizard displays the elements in a structured manner. The user can select which elements to add to the traceability link or can remove linked elements from the link (see Figure 31).

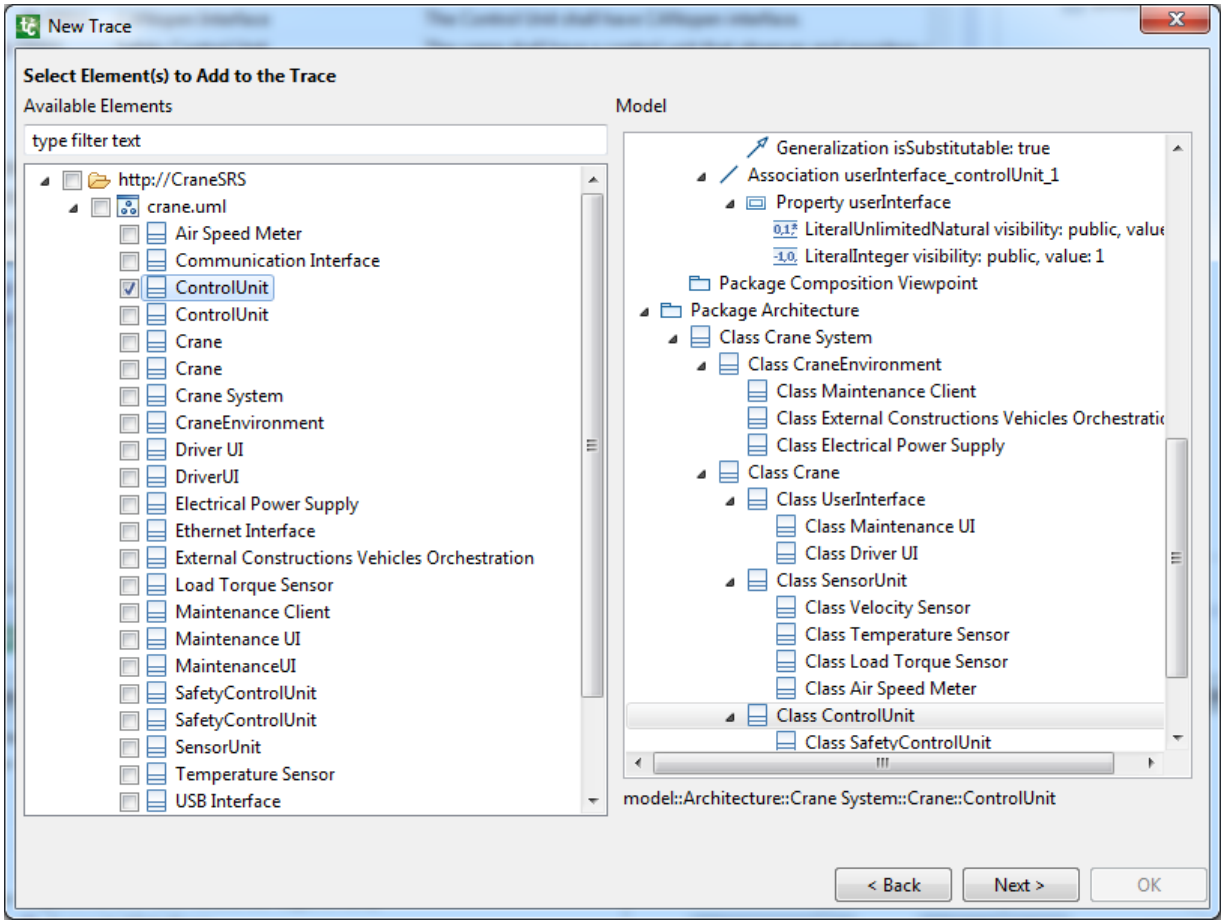


Figure 31 Managing Elements Linked by a Traceability Link

The viewer on the right side of the dialog shows the context of the selected model element, i.e. other model elements which are related to the element. This representation is supposed to be tool specific and ideally to be reused from the corresponding tool. The Traceino framework is extensible by tool specific implementations providing viewers to display models of a specific language. In case for UML for example, an implementation could render a UML class diagram containing a selected class element.

4. Steps 2 and 3 should be repeated for each trace reference of the traceability link type. When pressing the *OK* button of the wizard, the changes are applied to the traceability model. The *OK* button is enabled only if the traceability link is complete, i.e. if all its trace references link at least as much elements as the lower bound of the trace references indicates.

3.2.2 Create New Traceability Links

In order to create a new traceability link, please select a model element in a tool that is integrated with Traceino. If the Traceino framework is able to detect a traceability meta model including a traceability link type that can link elements of the type of the selected element, the *New…* button next to the combo box in the Traces View for selecting traceability links will be enabled (see Figure 32).

Figure 32 Button for Creating New Traceability Link

Clicking the button opens the Trace Properties Wizard explained in section 3.2.1.3 which allows creating new traceability links. Please follow the instructions of the wizard to model the traceability link as desired. When done, please click on the *OK* button of the wizard to store the traceability link and to close the wizard afterwards.

3.2.3 Modify Traceability Links

The *Edit…* button next to the combo box in the Traces View allows editing the properties of a traceability link, especially its linked elements (see Figure 31).

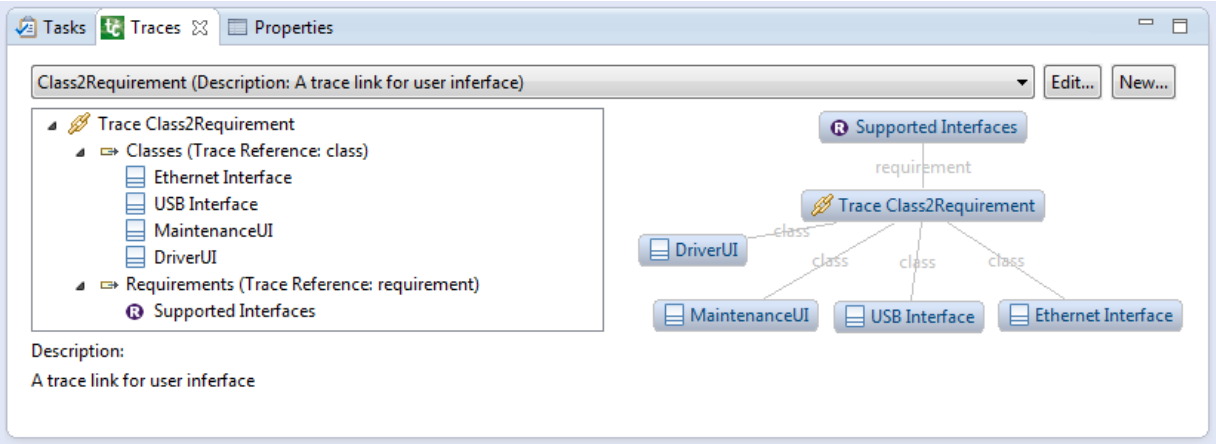


Figure 33 Button for Editing a Traceability Link's Properties

The button is enabled only if a traceability link is selected. In the most common setup of the Traceino specific views, this requires to select an element in the Traced Elements View. If the selected element is linked by multiple traceability links, the user is also required to choose the appropriate traceability link in the corresponding combo box in the Traces View (see section 3.2.1.2).

3.2.4 Delete Traceability Links

In order to delete a traceability link, please open the context menu of the top level element in the hierarchical representation of the link and select the option “Delete Trace…” (see Figure 34).

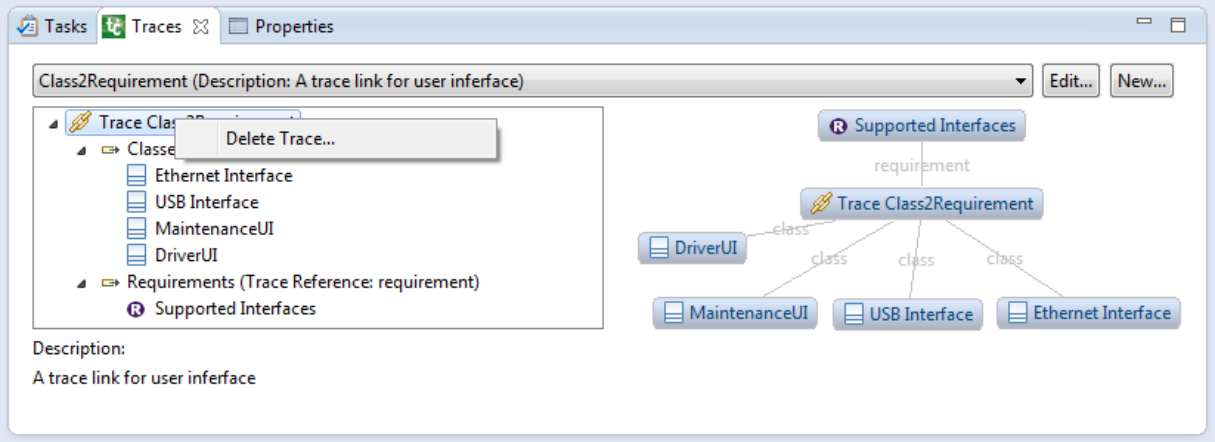


Figure 34 Deleting a Traceability Link

After having confirmed the decision, the traceability link will be deleted.

3.2.5 Modelbus® Manager Integration

Since release 1.9.8 the ModelBus Server (see http://www.modelbus.org/modelbus/index.php/downloads for more information) is shipped with an adapter for the ModelBus Manager web application (please refer to the ModelBus Users Guide for more information about the ModelBus Manager application).

The adapter allows managing traceability links from a central point of view. First, you have to switch to the *Traceability* perspective of the application which offers the Traceino specific views like the *Linked Elements View* (see section 3.2.1.1) and the *Traces View* (see section 3.2.1.2). Therefore, please click on the *other…* button in the upper left corner of the application window. This will open a dialog for opening a perspective (see Figure 35).

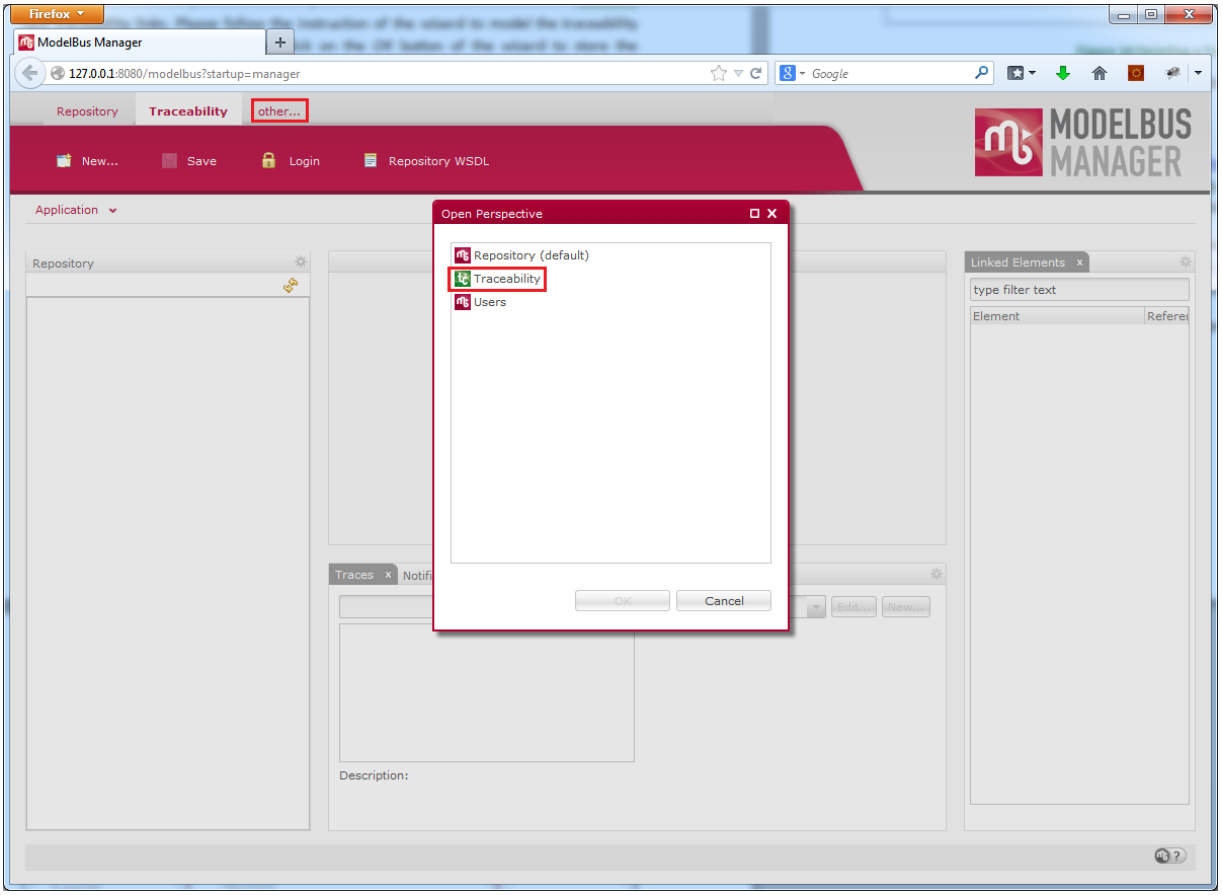


Figure 35 Opening the Traceability Perspective in ModelBus® Manager

In order to open a model within the ModelBus Manager tool, you are required to authenticate yourself. Please see the ModelBus® user guide to see how to accomplish this. When you have opened a model, please click on the “Tree” tab in the model viewer to show the model in a generic model tree viewer (see Figure 36).

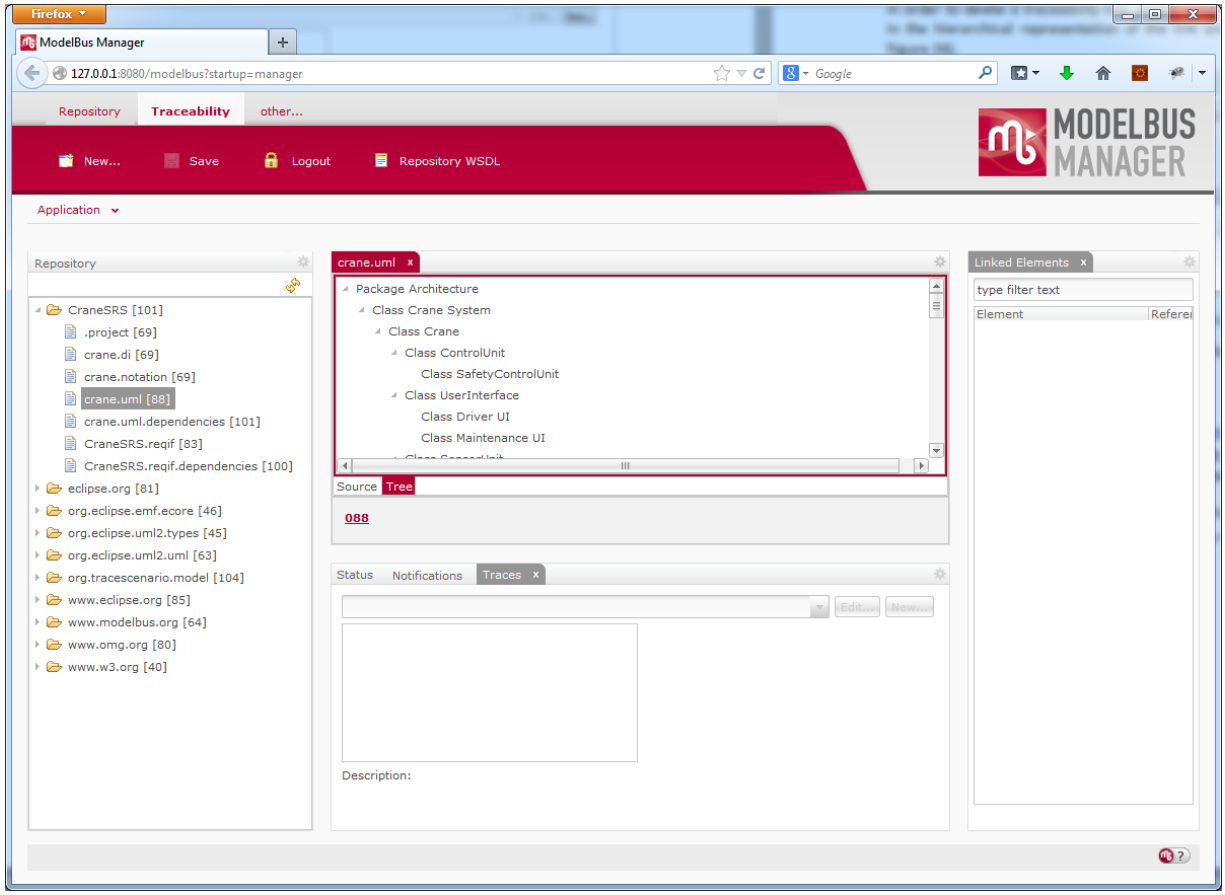


Figure 36 Model Tree Viewer in ModelBus® Manager

When selecting a model element within this tree viewer, the Traceino specific views, i.e. the *Linked Elements View* (see section 3.2.1.1) and the *Traces View* (see section 3.2.1.2), will show the model elements linked to the selected element and the properties of a traceability link, respectively. The handling of the views is identical to the workflow described in the previous sections.